Applied Computer Letters (ACL)

DOI: *http://doi.org/10.7508/acl.01.2018.05.10*

ISSN: 2377-6242 (Print)
ISSN: 2377-8156 (Online)

ARTICLE

A RECONFIGURABLE ACCELERATOR COMBINING FLEXIBILITY AND HIGH-ENERGY EFFICIENCY

Natasha Cattanach*

School of Computer Science, The University of Adelaide, Adelaide 5005, Australia
****Corresponding Author E-mail: Cattanach@gmail.com***

ARTICLE DETAILS

Article History:

Received 3 February 2018
Accepted 2 March 2018
Available online 29 March 2018

ABSTRACT

In this study, we proposed a novel dynamically reconfigurable accelerator "DYNaSTA". The DYNaSTA accelerator is a restricted dynamically reconfigurable accelerator composed of dynamically reconfigurable data paths called DYN and a static ALU array called STA, and we process the hot path of the program on behalf of the base processor. The STA computes the instructions in parallel, and DYN is dynamically reconfigured to solve the change in the operand dependency due to branch instructions. We designed the proposed DYNaSTA accelerator to operate at a clock frequency of 100 MHz using UMC 0.18 µm process, and simulated power consumption and measured the fabricated chip. Through the experiment, we obtained the results that power consumption reduced from 69% to 86% and energy efficiency improved from 4.5 times to 13 times. Therefore, the proposed DYNaSTA accelerator was proved to be a reconfigurable accelerator combining flexibility and high-energy efficiency.

KEYWORDS

Embedded microprocessor, accelerator, digital circuit, context controller, architecture.

1. INTRODUCTION

The overwhelming trend toward Internet of Things explains why low-energy embedded microprocessors (EMPs) are becoming increasingly important. Sources of energy inefficiency in EMP architectures are fairly well understood: the need to 1) fetch/decode every instruction from memory; 2) write/read register files to acquire/store operands per every instruction, and 3) clock numerous numbers of F/Fs for pipelining multiple instructions on a data path. The power consumption generated by these factors is not directly involved in computation [1].

Among the power consumption of general EMP, the proportion occupied by the ALU responsible for computation is approximately 10%, and the remaining 90% is occupied by redundant power irrelevant to computation (Figure 1). That is, by reducing such redundant power, we can improve the power efficiency of the EMP without degrading the computing performance. Thus, we may choose to "statically" map those instructions in heavily executed "recursive codes" to an array of ALUs prior to their execution. By running the codes only as combinatory data paths with no registers, 1)-3) redundancies can be drastically reduced [2-4].

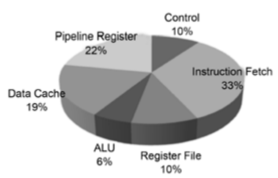

Figure 1: Example of EMP power consumption breakdown.

Although this "reconfigurable accelerator" solution looks straightforward and attractive, there is an inherent drawback: it is hard to cope with complex control flows (i.e., lots of branches) typically in embedded applications, which explain why previous proposals have focused on simple code segments that do not have a branch. Green Droid is a configurable processor for mobile devices with Android OS. The processor improves power efficiency by processing the hot path (most recursive code) of Android OS in hardware, but it has no versatility to other operating systems. ADRES is a processor in which dynamically reconfigurable function units (FU) are coupled to a very long instruction word (VLIW) processor. The processor improves performance by complementarily computing the hot path with the VLIW processor and FU array. However, the FU array cannot handle hot paths including multiple branch instructions, causing a decrease in energy efficiency. CMA is a reconfigurable processor with a processing element (PE) array consisting only of combinational circuits and it can be customized but cannot be dynamically reconfigured during execution [5, 6]. Therefore, although CMA is superior to conventional dynamically reconfigurable processors such as MuCCRA and DRP in power consumption, it has low flexibility and requires an external controller in order to execute a large-scale program.

As mentioned example above, the conventional reconfigurable processor is unable to combine power efficiency and flexibility. Based on this observation, we recently proposed an abstract architecture for achieving both energy efficiency and versatility in control-rich embedded applications. The architecture we proposed consists of a static arithmetic logic unit (ALU) array without registers and data paths that contains dynamically reconfigurable switches and registers, where the ALU array improves power efficiency and dynamically reconfigurable data paths ensure versatility. The contribution of this paper is to materialize the concept into executable micro-architecture, design/verify it in a silicon chip, and evaluate its energy efficiency.

In Section 2, we describe the architecture of the proposed DYNaSTA accelerator. The accelerator consists of a static data path, a dynamically reconfigurable data path, and circuits for controlling them. In Section 3, we show the simulation results for the DYNaSTA accelerator and the measurement results of the fabricated chip. The processor with the DYNaSTA accelerator showed reduced power consumption by 69% to 86% compared to general processors. In Section 4, we will summarize the study.

2. ARCHITECTURE

The key innovation in our proposal, a DYNaSTA reconfigurable accelerator, shown in Figure 2, is to combine two distinctive array structures different in nature, namely, a dynamic operand forwarding matrix (DYN) and a static ALU array (STA). STA computes an instruction sequence in parallel and plays a key role in achieving high-energy efficiency, where DYN is dynamically reconfigured while the accelerator is running and plays a key role to achieve versatility.

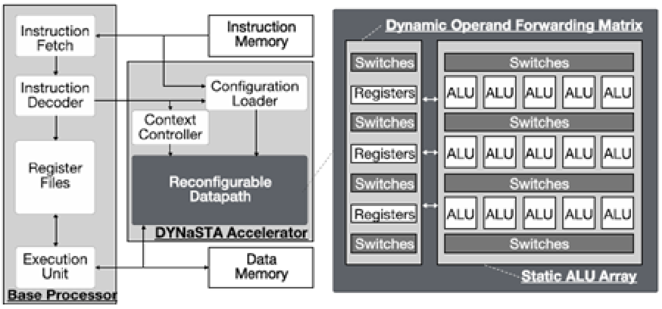

Figure 2: The reconfigurable datapath in the proposed DYNaSTA accelerator consists of a dynamic operand-forwarding matrix and static ALU array.

The DYNaSTA accelerator executes instructions by the method shown in Figure 3. When an instruction sequence to be executed by DYNaSTA is extracted, each instruction is mapped on the STA based on the data flow between the instructions, that is, the operand dependency, and the STA operates the instructions in parallel. If the data flow changes during operation because of the execution of a branch instruction, the switches of DYN are dynamically reconfigured and an appropriate data flow is constructed.

In the following subsection, we will describe in detail the architecture of each circuit included in the DYNaSTA accelerator.

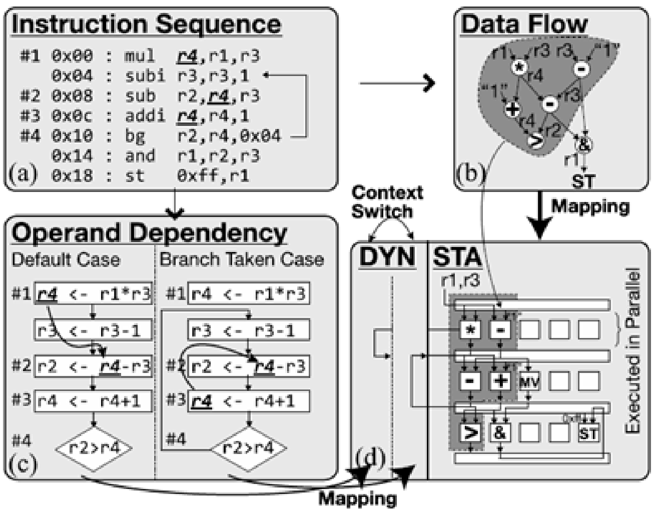

Figure 3: Code mapping policy: (a) an example code, (b) extracted data flow, (c) extracted operand dependency, and (d) mapping on DYN and STA.

2.1 Static ALU array

STA features a non-fixed number of stages, where each stage has several ALUs sharing a set of source/destination lines (Figure 4). To reduce the number of switches, hence improving energy efficiency, only parallel instructions are mapped onto a same stage, where branch/jump and load/store instructions go to the first and last ALUs, respectively (Figure 3(b) and Figure 3(d)). The instructions dependent on preceding ones are mapped onto the next stage. Conditional execution is supported for discarding short forward branches. An appropriate number of STA stages is dependent on the sizes of the target codes, whereas that of ALUs per stage will range from 2 to 8, as in superscalar/VLIW architectures. Note there are no registers and hence no clocks in STA.

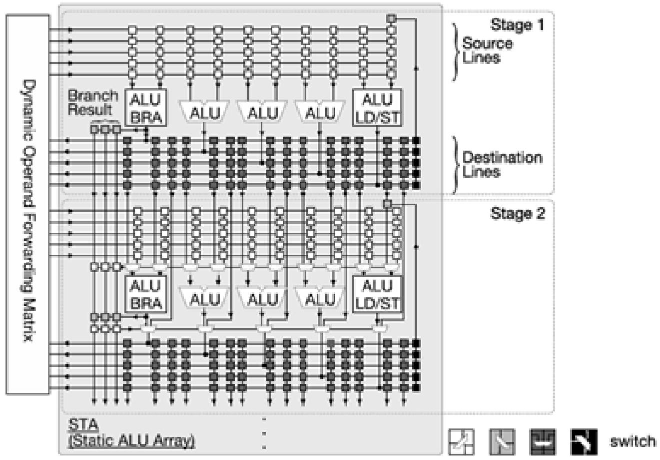

Figure 4: Block diagram of static ALU array (STA).

The difficulty in serving branches in a reconfigurable accelerator lies in that their outcome can never be known a priori: for example (Figure 3(c)), the “r4” operand in #2 may be produced by #3 instead of #1 when the #4 branch is taken. Efforts to accommodate this dynamic nature in ALU arrays such as STA unavoidably degrade its simplicity and regularity, hence incurring energy inefficiency.

2.2 Dynamic operand-forwarding matrix

DYN is a multi-context, bidirectional operand-forwarding matrix for solving this difficulty: it is dynamically reconfigured only when operand dependencies among instructions are altered on a branch (Figure 3(c) and Figure 3(d)). DYN is composed of temporary registers for storing operand values of each instruction and a large number of switches, as shown in Figure 5. When the data flow of the program transits while the accelerator is running, the switches are dynamically switched and appropriate data flow is constructed.

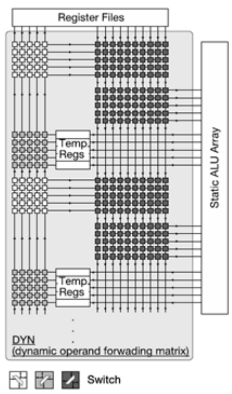

Figure 5: Block diagram of dynamic operand forwarding matrix (DYN).

Figure 6 represents an example in which fibonacci, used as one of the benchmark programs in the evaluation, is mapped to DYNaSTA. In Figure 6, it is shown that the datapath on DYN changes according to each context, and the appropriate operands are forwarded to the STA. Keeping power-consuming dynamic reconfiguration away from the massive ALU array (and leaving it static) is a key for achieving energy efficiency in DYNaSTA architecture.

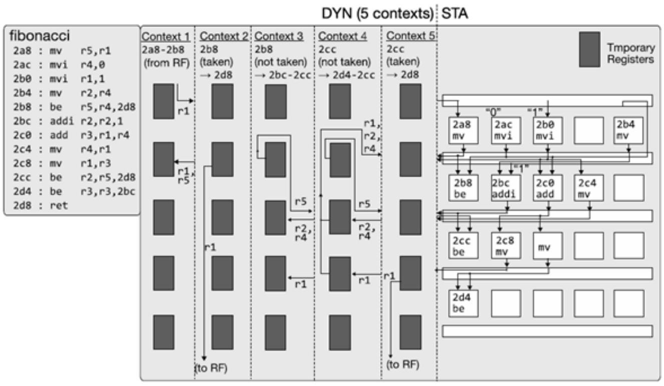

Figure 6: Assembly code of fibonacci and its mapping on DYNaSTA. When the context of the program transitions, DYN is dynamically reconfigured.

2.3 Context controller

The context controller shown in Figure 7 controls the transition of the context during execution. As the base processor starts executing the program, functions or subroutines processed by DYNaSTA are loaded from the instruction memory, and information of each context contained in them is stored in the configuration memory and the context memory. The context information is embedded in the executable file by recompiling the original executable file with the dedicated compiler; the function to be accelerated is also decided.

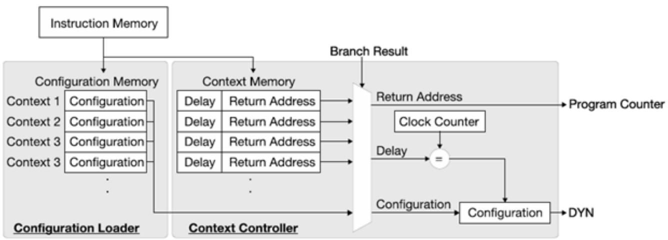

Figure 7: Block diagrams for configuration loader and context controller. DYN is dynamically reconfigured by these circuits.

The context information includes DYN's switch configuration information (configuration), the number of clock cycles required to execute the context (delay), and a return address. When the next context is selected according to the results of the branch instruction and the delay of the previous context is equal to the value of the clock counter (meaning that the previous context has been properly executed), the next context configuration information is sent to DYN. After DYNaSTA finishes executing the function, the base processor resumes program execution from the address of the instruction memory specified by the return address.

2.4 Overall architecture

We designed an EMP with this DYNaSTA accelerator into silicon (Figure 8). The base EMP is Mico32, which is chosen because of its typical RISC architecture and open-source RTL code. By treating "recursive codes" that are mapped onto DYNaSTA as subroutines, the read/write path between Mico32's RF and DYN only needs to cover its arguments portion (four registers, Figure 8).

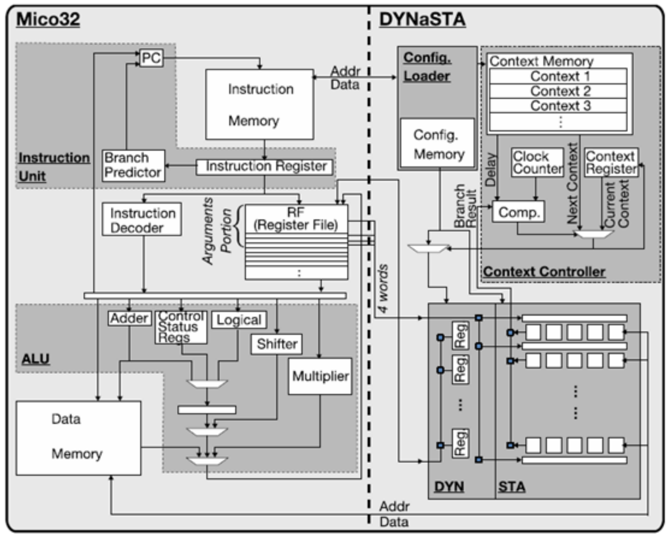

Figure 8: Tight integration of Mico32 (base EMP) and the DYNaSTA accelerator.

3. EVALUATION

3.1 Instruction-Level parallelism

Before simulating power consumption, we analyzed the optimal number of ALUs included in one stage of STA. If there are numerous unused ALUs, they generate unnecessary static power; in contrast, if there are only a few ALUs, instruction-level parallelism is reduced and computing performance is degraded. Therefore, we examined the relationship between the ALU occupancy and instruction-level parallelism through some programs containing many instructions from the benchmark set employed in the power-consumption simulation.

Figure 9 represents the result, in which the solid line represents the ALU occupancy, the dashed line represents the instruction-level parallelism, and the x-axis represents the number of ALUs per stage. The ALU occupancy depends linearly on the number of ALUs, whereas the instruction-level parallelism for crc32 and sbox is constant regardless of the number of ALUs. However, the instruction-level parallelism for sepia filter decreases when the number of ALUs is four. Therefore, we set the number of ALUs per stage to be five.

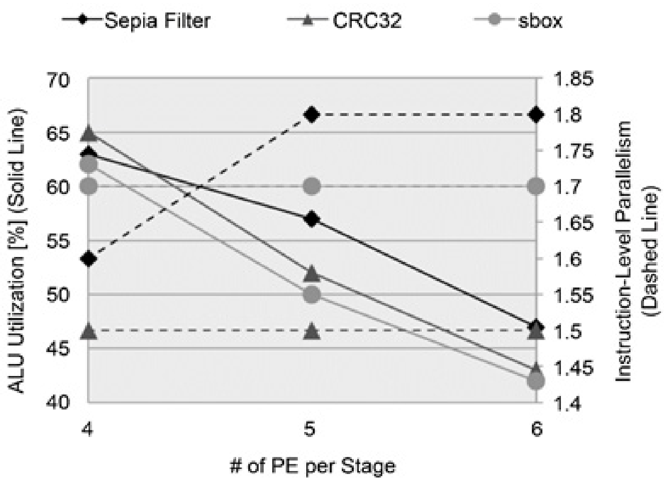

Figure 9: ALU utilization and instruction-level parallelism.

3.2 Power simulation

Then, the number of stages of the STA is set to 10, the performance and power consumption of the DYNaSTA accelerator were evaluated using sample applications (Table 1) based on the synthesized netlist. Figure 10 is a comparison of the power consumption when Mico32 and DYNaSTA execute the hot path of each application, that is, the most recursive code. As shown in the figure, the power consumption reduced by 69% to 86% due mainly to discarded instruction memory access. While Mico32 sequentially reads instructions from the instruction memory during program execution, DYNaSTA accesses the instruction memory only when generating configuration information (configuration phase) and does not access it during execution (running phase). Therefore, the power consumption to access the instruction memory has been greatly reduced. Logic power consumption is also reduced, as shown in Figure 10, whose detailed breakdown is shown in Figure 11 for the case of fibonacci.

Table 1: Summary of sample applications.

Application	# of instructions	# of branches	ALU utilization [%]	# of contexts
fibonacci	12	3	24	5
sbox	25	2	50	5
crc32	18	2	36	5
sepia filter	22	1	44	3

From Figure 10 and Figure 11, it is clear that the 1) to 3) redundancies mentioned earlier were successfully removed. Since instructions are executed in parallel in STA, the proposed architecture not only reduces the power but also enhances the performance (Figure 12) at the same frequency (100 MHz). As a result, the energy efficiency was improved 4.5 to 13 times from Mico32 for these sample codes.

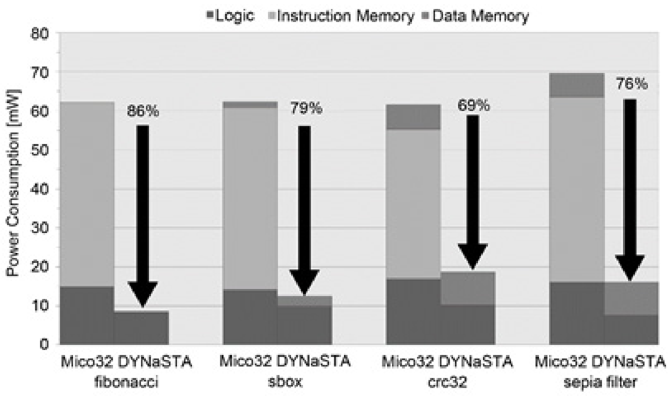

Figure 10: Mico32 vs. DYNaSTA: total power consumption of sample applications.

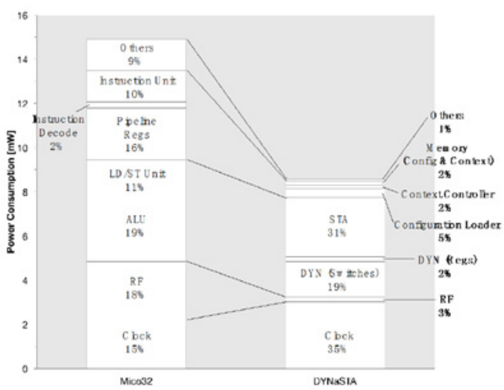

Figure 11: Mico32 vs. DYNaSTA: logic power consumption (fibonacci).

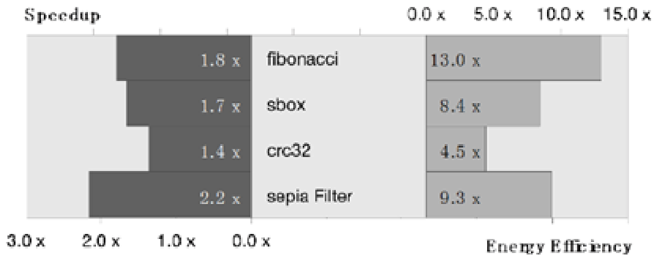

Figure 12: DYNaSTA/Mico32 performance and energy efficiency improvement.

3.3 Measurement of fabricated chip

We fabricated the proposed DYNaSTA using a UMC 0.18 µm process (see Figure 13 and Table 2). Because of the area constraint, four STA stages were implemented. The register file is originally installed on Mico32, we implemented it on DYNaSTA because we only designed the accelerator in this study. Although the size of DYNaSTA is very small, extending it is quite straightforward because of its regular array structure.

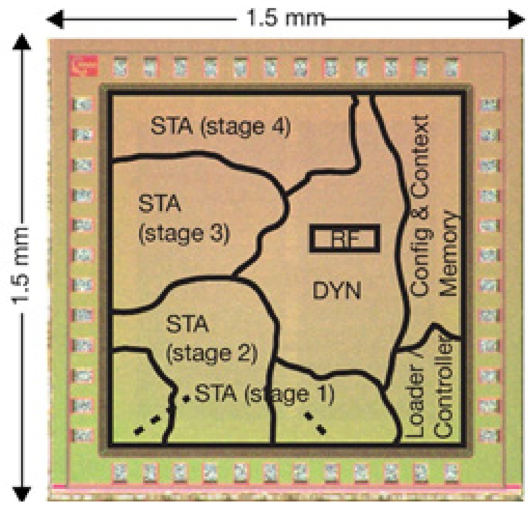

Figure 13: Chip micrograph of proposed DYNaSTA (four stages).

Table 2: Chip specifications.

Technology	UMC 0.18 µm 1P6M CMOS
Package	48-pin DIL
Die area	1.5 mm × 1.5 mm
Gate count	86.5 K
Supply voltage	1.8 V core/3.3 V IO
Clock frequency	100 MHz
# of stages	4
Register file	32 bit × 4 word
# of ALUs/stage	5
# of contexts	6

We measured the power consumptions of the fabricated chip during the configuration and the running phases of the DYNaSTA with fibonacci. The experimental setup is shown in Figure 14 and Figure 15. We implemented Mico32 on the FPGA (field-programmable gate array) and sent the test vector and clock to the fabricated DYNaSTA chip. Since DYNaSTA require 3.3 V power supply for I/O and 1.8 V for core, we supplied each power to DYNaSTA using two power supply units. Then, we connected the power analyzer to the core power supply and measured the power consumption during running. Figure 16 shows the measured power consumption versus clock frequencies for both the configuration and the running phases. Because of the limitation of our FPGA-based power-measurement workbench, the maximum frequency for the measurement was 80 MHz. We then predicted the power consumption at 100 MHz by linear interpolation of the measured data.

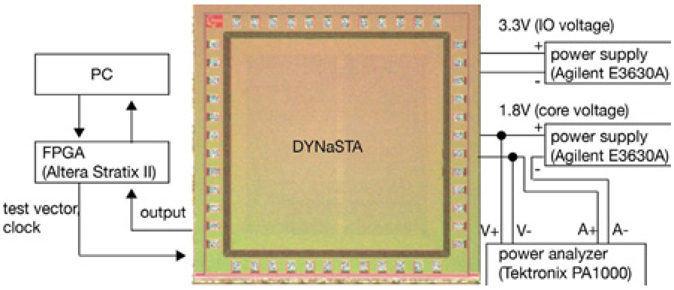

Figure 14: Experimental configuration.

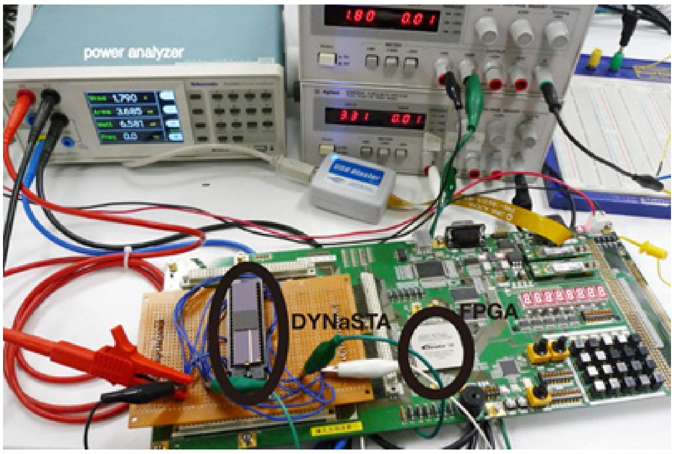

Figure 15: Photograph of our power-measurement workbench.

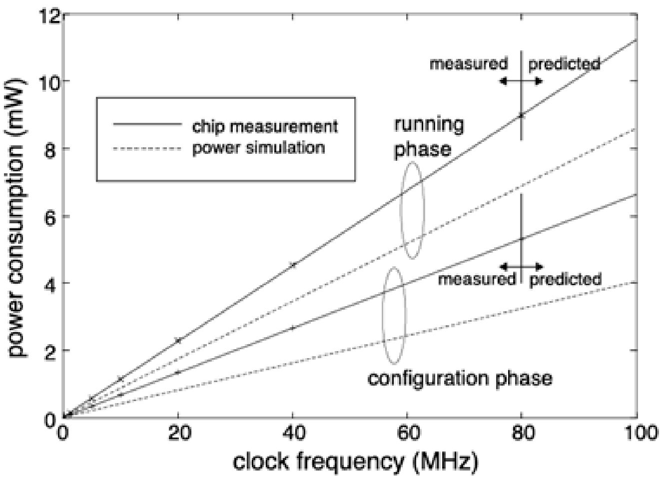

Figure 16: Measured power consumption vs. clock frequency for configuration and running phase.

Table 3 shows a comparison of the simulated and measured (and interpolated) power consumption for both phases at 100 MHz. We observed a slight mismatch of approximately 2.6 mW for both phases between the simulated and measured data. This mismatch resulted from circuit elements of the fabricated chip that were not included in the power simulation model, such as the Mico32 register file. Table 4 reveals the reasons for this energy efficiency: although DYNaSTA consumes ×18.5 more gates than Mico32, its average toggle rate is as low as ×0.06 of Mico32. Here, the average toggle rate represents the ratio of nodes that toggled synchronously with the rising (or falling) edge of the clock among all the nodes in the circuit per unit time. Specifically, gate-consuming STA features only a 1.8% toggle rate, which accounts for its relatively low power occupation in Figure 11.

Table 3: Comparison between simulation results and measurement results at 100 MHz.

	Configuration [mW]	Run [mW]
Simulation	4.03	8.57
Measurement (interpolated)	6.63	11.20

Table 4: Mico32 vs. DYNaSTA: gate counts and average toggle rates (fibonacci).

		Gate count [k gates]	Average toggle rate [%]
DYNaSTA	DYN	43.6	20.0
	STA	322.9	1.8
	Others	80.2	9.1
	All	446.8	4.9
Mico32		24.1	76.8
Ratio (DYNaSTA/Mico32)		18.48	0.06

4. CONCLUSION

In this study, we proposed a novel dynamically reconfigurable accelerator "DYNaSTA". The DYNaSTA accelerator is a restricted dynamically reconfigurable accelerator composed of dynamically reconfigurable data paths called DYN and a static ALU array called STA, and we process the hot path of the program on behalf of the base processor. The STA computes the instructions in parallel, and DYN is dynamically reconfigured to solve the change in the operand dependency due to branch instructions. We designed the proposed DYNaSTA accelerator to operate at a clock frequency of 100 MHz using UMC 0.18 μm process, and simulated power consumption and measured the fabricated chip. Through the experiment, we obtained the results that power consumption reduced from 69% to 86% and energy efficiency improved from 4.5 times to 13 times. Therefore, the proposed DYNaSTA accelerator was proved to be a reconfigurable accelerator combining flexibility and high-energy efficiency.

Filling a chip with simple, regular, and energy-efficient array like DYNaSTA can become an interesting solution in the "Dark Silicon" era (Figure 17). Here, existing domain-oriented low-power-circuit techniques such as DVFS and power gating can augment the architecture quite nicely. For instance, since only a few active stages propagate like a "wave" on the array, remaining numerous "silent" stages can be powered-off systematically to minimize the leak current (Figure 17). Our next challenges include enhancing DYNaSTA with such lowpower-circuit techniques as well as establishing code mapping SW.

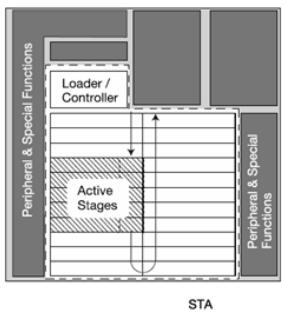

Figure 17: DYNaSTA SoC concept toward "Dark Sillicon" era.

REFERENCES

[1] Clark, L.T., Patterson, D.W., Ramamurthy, C., Holbert, K.E. An Embedded Microprocessor Radiation Hardened by Microarchitecture and Circuits. IEEE TRANSACTIONS ON COMPUTERS (2016) 65(2): 382-395.

[2] Kim, H., Sejun, K., Choi, J. Design and Implementation of Embedded System based on AM3359 Microprocessor(AM3359 마이크로프로세서 기반 임베디드 시스템 설계 및 제작). IEMEK Journal of Embedded Systems and Applications (2017) 12(2): 89-96.

[3] Park, S., Park, C.S. Design of Low-Gate-Count Low-Power Microprocessors with High Code Density for Deeply Embedded Applications. JOURNAL OF CIRCUITS SYSTEMS AND COMPUTERS (2017) 26(17501329).

[4] Sato, R., Hatanaka, Y., Ando, Y., Tanaka, M., Fujimaki, A., Takagi, K., Takagi, N. High-Speed Operation of Random-Access-Memory-Embedded Microprocessor With Minimal Instruction Set Architecture Based on Rapid Single-Flux-Quantum Logic. IEEE TRANSACTIONS ON APPLIED SUPERCONDUCTIVITY (2017) 27(13005054).

[5] Boussadi, M.A., Tixier, T., Landrault, A., Derutin, J. HNCP: A many-core microprocessor ASIC approach dedicated to embedded image processing applications. MICROPROCESSORS AND MICROSYSTEMS (2016) 47(SIB): 333-346.

[6] Lins, F.M., Tambara, L.A., Kastensmidt, F.L., Rech, P. Register File Criticality and Compiler Optimization Effects on Embedded Microprocessor Reliability. IEEE TRANSACTIONS ON NUCLEAR SCIENCE (2017) 64(81): 2179-2187.